Pathfinding in 3D Space
CHIA-MAN HUNG & RUOQI HE
Outline

- Introduction
- I. State of the art
- II. Algorithms
- III. Implementation in 3D space
- IV. Results
- Conclusion
Introduction

- Objective: Find the shortest paths efficiently in 3D space
- Applications: video games, drone navigation
I. State of the art

- Homeworld (1999):
  First famous real-time strategy game with movement in 3D space
1. State of the art

- Shortest paths in a graph
  - Dijkstra (single source)
    - $O((|V|+|E|)\log(|V|))$
  - Bellman-Ford (single source, weighted directed graph)
    - $O(|V||E|)$
  - Floyd-Warshall (for all pairs of vertices, weighted graph, no negative cycle)
    - $O(|V|^3)$
  - A* (single source, single destination)
    - $O(n)$, $n =$ length of the solution path $\Rightarrow O(|E|)$
I. State of the art

- 2D - exact
  - Visibility graph
  - Anya (2D grid)

- 2D - approximate
  - Waypoints
  - Navigation mesh + tunnel
  - Family of Theta*
Non-optimality

- Navigation mesh + tunnel

path found VS true shortest path
I. State of the art

- 3D surface - exact
  - Windows (Fast exact and approximate geodesics on meshes 2005 Surazhsky)

- 3D surface - approximate
  - Heat (Geodesics in heat 2013 Crane)
  - Fast-marching (1996 Sethian)
II. Algorithms

- World representation
  - Tetrahedralization
  - Convex decomposition
  - Grid
  - Octree
II. Algorithms

- A* (1968 Hart)
  
h admissible if no over-estimation and \( h(y) \leq h(x) + d(x, y) \)
II. Algorithms

- Theta* (2007 Nash)
II. Algorithms

- Lazy Theta* (2010 Nash)

```c
1 Main()
2   open := closed := \emptyset;
3   g(s_{start}) := 0;
4   parent(s_{start}) := s_{start};
5   open.Insert(s_{start}, g(s_{start}) + h(s_{start}));
6   while open \neq \emptyset do
7       s := open.Pop();
8       SetVertex(s);
9           if s = s_{goal} then
10              return "path found";
11           closed := closed \cup \{ s \};
12           foreach s' \in \text{neighbors}(s) do
13               if s' \notin closed then
14                   if s' \notin open then
15                       g(s') := \infty;
16                       parent(s') := NULL;
17                       UpdateVertex(s, s');
18                   return "no path found";
19               end
20               ComputeCost(s, s');
21                   if g(s') < g(old) then
22                       if s' \in open then
23                           open.Remove(s');
24                           open.Insert(s', g(s') + h(s'));
25                       end
26                   end
27               end
28               ComputeCost(s, s');
29                 /* Path 2 */
30                 if g(parent(s)) + c(parent(s), s') < g(s') then
31                     parent(s') := parent(s);
32                     g(s') := g(parent(s)) + c(parent(s), s');
33                 end
34               end
35               SetVertex(s);
36                 if NOT lineofsight(parent(s), s) then
37                   /* Path 1 */
38                     parent(s) := 
39                         \arg\min_{s' \in \text{neighbors}(s) \setminus \text{closed}} (g(s') + c(s', s));
40                     g(s) := 
41                         \min_{s' \in \text{neighbors}(s) \setminus \text{closed}} (g(s') + c(s', s));
42                 end
```
III. Implementation

- Octree construction
  - Triangle-cube intersection
  - Progressive octree
- Graph construction

Dual graph (not standard)  Edge-corner
III. Implementation

- Line of sight
  - Fast
  - Robust
III. Implementation

- Injection of source and destination
III. Implementation - Optimisation

- Avoid exhaustive search
  - Precompute the connectivity of the graph nodes
III. Implementation - Optimisation

- Multisource
  - Reuse information
III. Implementation - Extension

- Application in video games
  - Waypoints
  - Repulsive force
  - Replanning
## IV. Results

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Algorithm</th>
<th>distance</th>
<th>time cost</th>
<th>distance</th>
<th>time cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octree</td>
<td>A*</td>
<td>125.18%</td>
<td>1.6ms</td>
<td>124.56%</td>
<td>4.5ms</td>
</tr>
<tr>
<td></td>
<td>Theta*</td>
<td>101.58%</td>
<td>11.8ms</td>
<td>109.24%</td>
<td>53.0ms</td>
</tr>
<tr>
<td></td>
<td>Lazy Theta*</td>
<td>101.73%</td>
<td>6.6ms</td>
<td>109.48%</td>
<td>22.1ms</td>
</tr>
<tr>
<td>Progressive Octree</td>
<td>A*</td>
<td>125.19%</td>
<td>4.6ms</td>
<td>126.27%</td>
<td>5.8ms</td>
</tr>
<tr>
<td></td>
<td>Theta*</td>
<td>101.58%</td>
<td>21.5ms</td>
<td>101.63%</td>
<td>42.7ms</td>
</tr>
<tr>
<td></td>
<td>Lazy Theta*</td>
<td>101.39%</td>
<td>10.6ms</td>
<td>102.27%</td>
<td>18.1ms</td>
</tr>
</tbody>
</table>

Table 1: Comparison - A single wall - Left: source sparse/ Right: destination sparse

- Red: A*
- Green: Theta*
- Blue: Lazy Theta*
IV. Results

- **Red:** A*  
- **Green:** Theta*  
- **Blue:** Lazy Theta*

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Algorithm</th>
<th>distance</th>
<th>time cost</th>
<th>distance</th>
<th>time cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Octree</strong></td>
<td>A*</td>
<td>121.38%</td>
<td>1.5ms</td>
<td>121.03%</td>
<td>28.0ms</td>
</tr>
<tr>
<td></td>
<td>Theta*</td>
<td>104.59%</td>
<td>6.7ms</td>
<td>102.71%</td>
<td>236.1ms</td>
</tr>
<tr>
<td></td>
<td>Lazy Theta*</td>
<td>104.65%</td>
<td>4.2ms</td>
<td>102.34%</td>
<td>114.8ms</td>
</tr>
<tr>
<td><strong>Progressive Octree</strong></td>
<td>A*</td>
<td>127.43%</td>
<td>3.3ms</td>
<td>126.88%</td>
<td>55.6ms</td>
</tr>
<tr>
<td></td>
<td>Theta*</td>
<td>103.49%</td>
<td>6.3ms</td>
<td>101.23%</td>
<td>229.4ms</td>
</tr>
<tr>
<td></td>
<td>Lazy Theta*</td>
<td>103.68%</td>
<td>3.2ms</td>
<td>101.16%</td>
<td>108.8ms</td>
</tr>
</tbody>
</table>

Table 2: Comparison - A single sphere - Left: level = 7/ Right: level = 9
IV. Results

Red: A*
Green: Theta*
Blue: Lazy Theta*

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Algorithm</th>
<th>distance</th>
<th>time cost</th>
<th>distance</th>
<th>time cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octree</td>
<td>A*</td>
<td>3.2472</td>
<td>9.5ms</td>
<td>3.3302</td>
<td>5.3ms</td>
</tr>
<tr>
<td></td>
<td>Theta*</td>
<td>2.4108</td>
<td>23.9ms</td>
<td>2.4600</td>
<td>45.5ms</td>
</tr>
<tr>
<td></td>
<td>Lazy Theta*</td>
<td>2.4135</td>
<td>9.5ms</td>
<td>2.4592</td>
<td>16.3ms</td>
</tr>
<tr>
<td>Progressive Octree</td>
<td>A*</td>
<td>3.3949</td>
<td>14.1ms</td>
<td>3.3222</td>
<td>7.15ms</td>
</tr>
<tr>
<td></td>
<td>Theta*</td>
<td>2.4009</td>
<td>17.59ms</td>
<td>2.4158</td>
<td>43.1ms</td>
</tr>
<tr>
<td></td>
<td>Lazy Theta*</td>
<td>2.4057</td>
<td>7.78ms</td>
<td>2.4205</td>
<td>14.6ms</td>
</tr>
</tbody>
</table>

Table 3: Comparison - A complex scene - Left: edge-corner graph/ Right: dual graph
Non-optimality of Theta*
Demo!

- Demo!
  - Demo!
    - Demo!
      - Demo!
        - Demo!
Conclusion

- Exploration in a new domain
- Our proposition: Lazy Theta * + Progressive Octree + Edge-corner graph
- Possible Improvements
  - Distribution of computation at each frame
  - Other possibilities of h
  - Post-processing